Finance4J Library

Developer's Manual

Last Updated: 18Aug2012

Table of Contents

- Servoy Plug-in
  - General
    - Developer System Requirements
    - Installation
    - Activation
    - Removal
  - Usage
    - Deployment
    - Connecting to QuickBooks
      - Backwards Compatibility
      - Local Connection
      - Remote Connection (using Finance Server)
      - Sessions
    - Methods
      - Request Methods
      - Type Builder Methods
      - Dates and Times
      - Other Methods
      - Error Handling
    - Advanced
      - XML
- Finance Server
  - General
    - System Requirements
    - Installation
    - Removal
  - Setup
    - General
      - Server Port
      - Persistence
      - Aliases
    - Users
    - Security
      - IP Filtering
- Contact Us

Servoy Plug-in

General

Developer System Requirements

- **Windows** for local QuickBooks connection
- **Windows, Linux**, or **Mac OS X**, for remote QuickBooks connection
- **Servoy (3.x or higher)**
- **QuickBooks**
- **Java**

Installation

To install the plug-in, first install the two included dependencies:

- QBXML Redistributable Package (included)
- Finance4J Redistributable (included)

Please note: Clients accessing solutions powered by the Finance plug-in will also need these dependencies installed on their end.

Once you have installed the dependencies, run the plug-in installer jar file. Follow the instructions carefully, as you'll need to chose your version of Servoy during the installation. Choose the directory you installed Servoy into.

Once you have the plug-in installed into your Servoy directory, start Servoy Developer.

If a license dialog does not come up, you may need to upgrade Java.

Activation

After you have installed the plug-in and started Servoy Developer, a dialog box will appear prompting you for your license details. If you do not know this information, refer to the e-mail you received when you purchased a license or downloaded a trial.

If you receive an "Authentication Successful" message, you are ready to start using the plug-in. If you are behind a proxy or cannot otherwise connect to the Internet, choose "Manual Activation" instead. This option is not available for trial users.

This process only activates the plug-in for developer usage, you will still need to use run-time activation when deploying (see *Deployment* under *Usage* below).

Removal

To remove the plug-in, first ensure Servoy is not running. Navigate to your Servoy installation folder. For Servoy 4+ users, open the application_server folder. Inside the *plugins* folder, delete the following files and folders:

- Finance.jar
- Finance.jnlp
- Finance *(folder)*

To completely remove all activations and settings, navigate to your home directory (under Windows this is C:\Documents and Settings\[Your Username]\ or C:\Users\[Your Username]\). Open the ".servoy" folder. This may be hidden on non-Windows systems. Delete the Finance folder, if it exists.

Usage

Deployment

In order to deploy, your code needs to call the *unlockRuntime(...)* method before any other plug-in methods. To retrieve your runtime code, log into your account at www.prolificaxis.com. Deployment is not available to trial users.

```
plugins.Finance.unlockRuntime("YOUR RUNTIME CODE HERE");
```

Connecting to QuickBooks

Backwards Compatibility

The Finance plug-in targets QBXML 7.0. However, in order to send messages to older version of QuickBooks (such as 2007 or below), be sure to set the QBXML version lower, for maximum compatibility. You may lose some newer features doing so, but it will allow for your solution to have backwards compatibility. You can change this property at any time:

```
plugins.Finance.QBXMLVersion = "6.0"; // QB 2007
```

Local Connection

To open a connection with QuickBooks installed on the same machine, use the *openLocalConnection* method:

```
plugins.Finance.openLocalConnection("FinancePlugin","FinancePlugin");
```

You will need to have QuickBooks running the first time you connect in order to allow the connection.

You may pass in the name of your application or something like the above example. Local connections only work in Windows. However, you can achieve similar behavior on other operating systems by running QuickBooks in a Windows virtual machine and using remote connection to connect with it.

To determine if the connection succeeded, use the methods listed in *Error Handling.*

Connections automatically close when the solution exits. However, to close a connection manually, call:

```
plugins.Finance.closeConnection();
```

Remote Connection (using Finance Server)

To open a connection with a QuickBooks solution running on another machine, the other machine must have the Finance Server set up and installed. See the *Finance Server* portion of the manual for details on configuration and setup.

Opening a remote connection is similar to a local connection, but differs in that it also requires a hostname, port, user name and a key. The username and key fields must contain only letters and numbers, and no special symbols. The method returns a boolean indicating success. If false is returned, an error may have occurred.

- boolean openRemoteConnection(String hostname, int port, String appID, String appName, String user, String key)

Examples:

```
plugins.Finance.openRemoteConnection("192.168.1.101", 10230,
"FinancePlugin", "FinancePlugin", "testuser", "12345");
```

The user name and key used must be added to the server beforehand, and your IP or IP range may need to be white-listed depending on the server configuration.

If the connection is a success, you can proceed with opening a session as you would with a local connection.

Sessions

Once you have established a connection, you can open a session by calling:

```
plugins.Finance.openSession("c:/test.qbw");
```

The first parameter (String) is the path to your qbw file you wish to connect to.
The optional second parameter (int) is an integer representing the open mode used for the session. Valid values for this parameter are:

- 1: Single User Mode (default)
- 2: Multi-User Mode
- 3: Don't Care

If you are using a remote connection, the Finance Server may change this depending on your server settings.

Once a session has been opened, you are ready to start sending data to QuickBooks.

Sessions automatically close when the solution exits. However, to close a session manually, call:

```
plugins.Finance.closeSession();
```

Methods

Request Methods

The Finance plug-in works by providing a way to generate specific QBXML requests and get that data to QuickBooks. Each QBXML request takes the form of a method in the plug-in, with a capital letter, such as *InvoiceAdd*. The parameters of each method are different.

The methods return a String that can be then given to the *request(...)* method, which will attempt to send QuickBooks the data provided according to the current connection and session information.

What each method requires can be determined by the names of the parameters. The parameters types can either be *qbXMLInfo* or another nullable type (String, Integer, etc.). A *qbXMLInfo* type means that the type needs to be built with a type builder method (see *Type Builder Methods*). Some request may require a date or time type (see *Date and Times below* for correct formatting).

Not all parameters are required. Any parameter can be left null, but QuickBooks may reject it (by returning an error) if it does not have enough information. What each query needs is the same as what would be expected if performing the action inside of the QuickBooks application itself.

Since QuickBooks allows many different options when specifying parameters, you may find you are leaving a large number of the parameters as null, which is normal.

Type Builder Methods

The type builder commands are prefixed with *XML_*. These are needed to build specific types required by the request methods mentioned above (or other type builder methods).

If you need to specify multiple type builder commands together inside a single

parameter, such as with an Invoice Line Item, you may append them together with the list(...) method (see *Other Methods).*

Dates and Times

To add a **date** or **time**, you can use QuickBooks dates and times format as a String inside your requests. Simply provide a string in the format *YYYY-MM-DD*. To give an exact time, use the format *YYYY-MM-DDTHH-MM-SS* where *T* is a separator between the date and time.

Other Methods

- String getCurrentCompanyFileName()
  Returns the current company file name directly from QuickBooks, if there is one. This may be blank if there is no connection. This method may generate an error.

- qbXMLInfo list(...)
  Appends multiple type builder methods into one entry.

Error Handling

The Finance plug-in has two separate forms of error-handling.

Standard error handling:

- *boolean hasError()*
  Returns whether or not the plug-in had an error.
  It is recommended to check this after every plug-in method call.

- *String errorString()*
  Returns a String detailing the specific error, otherwise returns empty string.
  This is intended for logging purposes.

- *void clearErrors()*
  Clears the error messages. Does not clear Quickbooks errors.

To see Quickbooks' errors directly, and to get any relevant error codes:

- *String getQBError()*

- *int getQBErrorCode()*

Advanced

XML

The Finance plug-in allows for developers to send their own QBXML data to QuickBooks. This is possible by building the XML requests manually and sending them but as a parameter to the *request()* method.

Finance Server

General

System Requirements

- **Windows**
- **QuickBooks**
- **Java**

Installation

To get started using the Finance Server, extract the server and accompanying files to a directory of your choice. Run the server once, and close it. This will automatically create the directories needed to configure it. Now you are ready to set up your server (see *Setup*).

Removal

To remove the Finance Server, simply delete the files you extracted when installing. To completely remove your settings (optional), navigate to your home directory (under Windows this is C:\Documents and Settings\[Your Username]\ or C:\Users\[Your Username]\). Delete the ".finance4j" folder.

Setup

General

The general configuration settings for the server are located in *config.ini.* This file is located inside the .finance4j settings folder (under Windows this is C:\Documents and Settings\[Your Username]\.finance4j or C:\Users\[Your Username]\.finance4j).

Server Port

The default port for the Finance Server is *10230*. You can change it by using the server port option as shown in the snippet below:

```
[server]
port=10300
```

Persistence

The following snippet, to be placed in *config.ini*, enables the persistent database

feature:

```
[server]
persist=true
```

This keeps connections and sessions using the Finance Server in an “always open” mode, in order to speed up multiple connections by not requiring the reopening connections to the same QuickBooks files. In most cases, persist mode will be more efficient. The downside is that all of your connections will need to use the same *appID* and *appName* to be able to connect simultaneously, as well as the same *filename* and *open mode* parameter.

Even though persistence may avoid actually reopening connections and sessions on the server side, you will still need to call *openConnection* and *openSession* as you normally would with a non-persistent remote connection.

Aliases

The Finance Server uses aliases for easy redirection to a specific company file. This prevents needing to use the full path name to access company files when using remote connection. You can set up as many aliases as desired. Simple add entries using the following format to *config.ini,* as shown below:

```
[alias]
test_company=c:/path_to_company_file/company.qbw
```

Use forward slashes for paths.

The follow alias will allow clients to connect specifying “*test_company*” instead of the full path to the company file. The full path name will still be retrievable on the client end using the *getCurrentCompanyFileName()* method.

Users

To set up your users and associated keys, open the file called *keys.ini.* This file is located inside the .finance4j settings folder (under Windows this is C:\Documents and Settings\[Your Username]\.finance4j or C:\Users\[Your Username]\.finance4j). If it is not there, you can create it.

To add a user to the system, simply create a new section. The name of the section is the user name, and the value of ‘key’ for that user is the key to use for that user. The below example shows a possible keys.ini for a user called *JohnSmith* and a key *12345.*

```
[JohnSmith]
key=12345
```

Security

IP Filtering

Another layer of security can be achieved through *IP filtering.* This allows you to white-list or black-list certain IPs or IP ranges and their access to the server. You can configure this through *config.ini*.

IP filtering is also compatible with IPv6.

The follow snippet is an example of allowing only a certain range of IPs to connect to the Finance server. Note the use of a wild-card, and listing multiple IPs separated by commas.

```
[security]
type=whitelist
ips=192.168.1.*,127.0.0.1
```

The next snippet shows an example of a black list, which allows all IPs except those listed. Note the use of ranges.

```
[security]
type=blacklist
ips=0-126.*.*.*,128-191.*.*.*,193-255.*.*.*
```

If no security type is given, the default behavior allows all IPs to connect, provided they have the correct user-key combination.

Keep in mind these methods are provided as convenience only, and that you are still responsible for securing your server and your data, as well as complying with any applicable legal security regulations.

Contact Us

Prolific Axis, LLC
Website: http://www.prolificaxis.com
Support: http://www.prolificaxis.com/support
E-mail: prolificaxis@prolificaxis.com